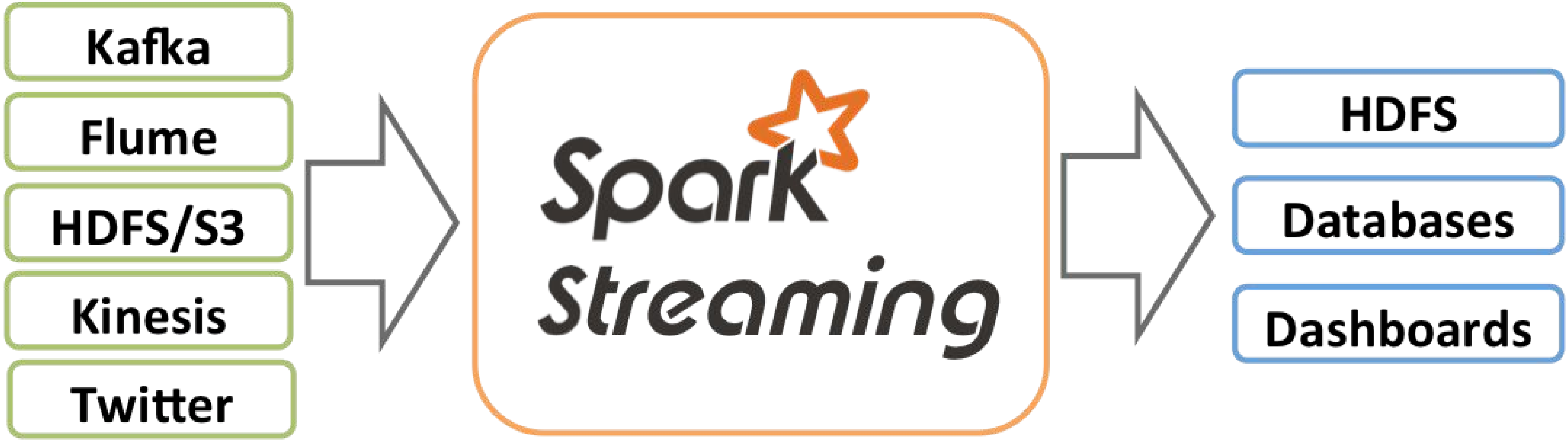
Spark Streaming

Intro - 1/16/2017

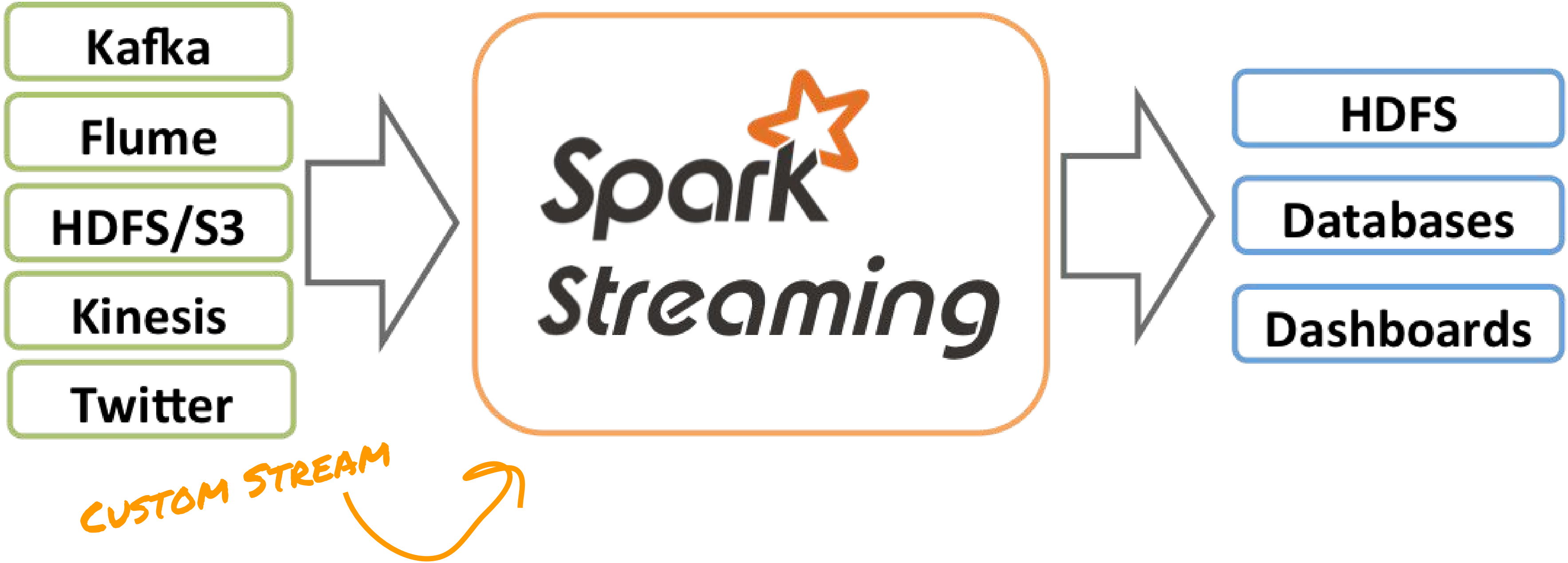
Agenda

- Place of Spark Streaming in the ETL pipeline and rest of the spark universe
- DStreams
    - batches, microbatches, RDDs, transformations
- Windowed computations
- Streaming Context via some code

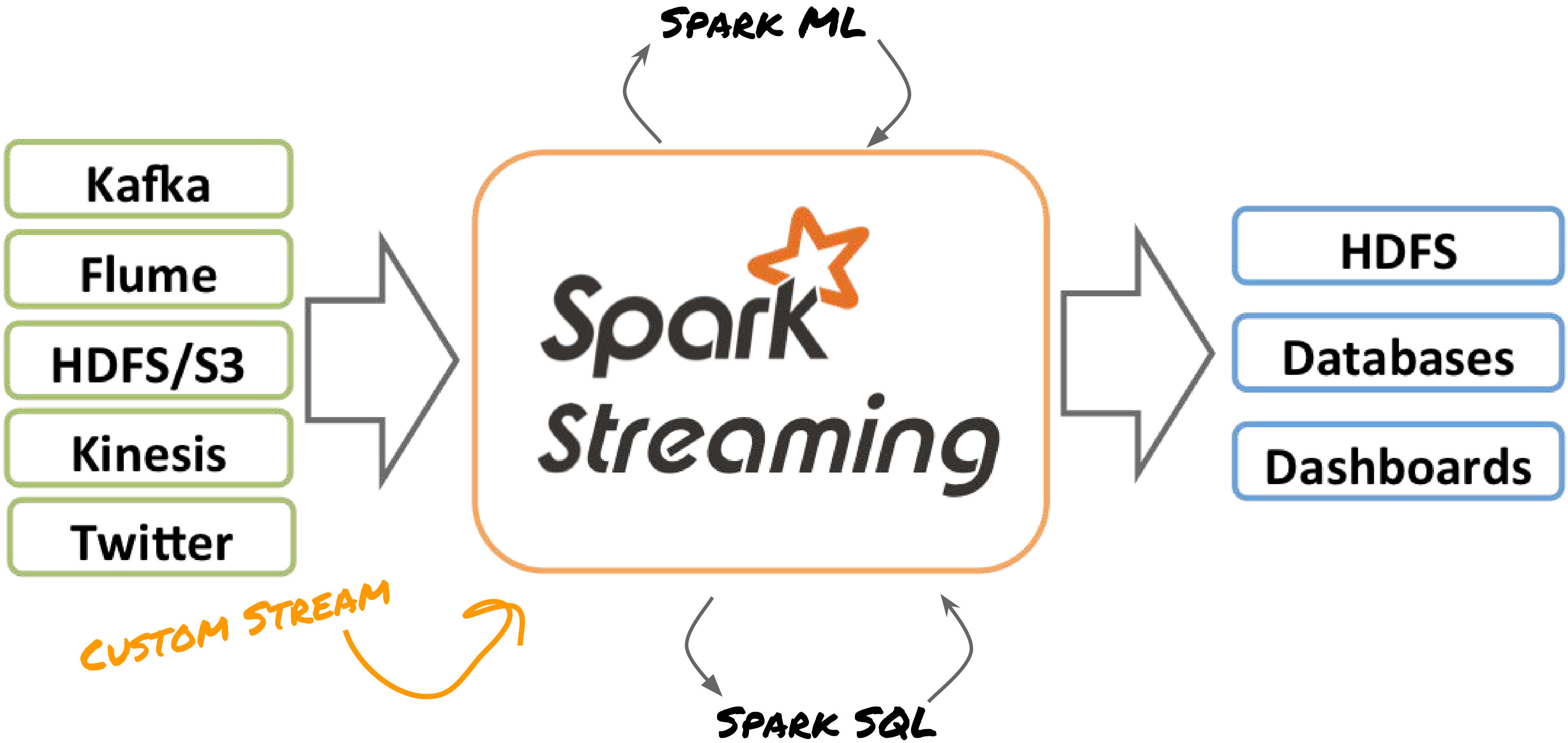
E. Transformation. L.



E. Transformation. L.



E. Transformation. L.



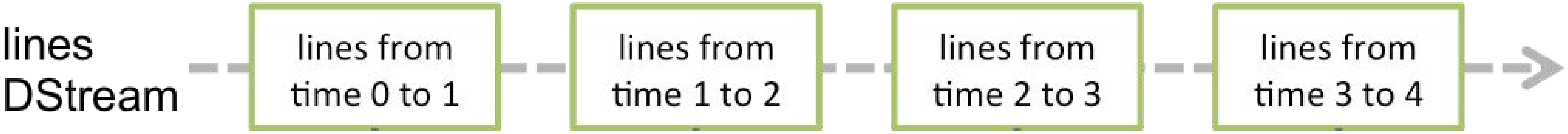
DStream

Discretized Stream

Stored as a continuing series of RDDs

Every DStream operation, triggers RDD transformation

DStreams



lines DStream

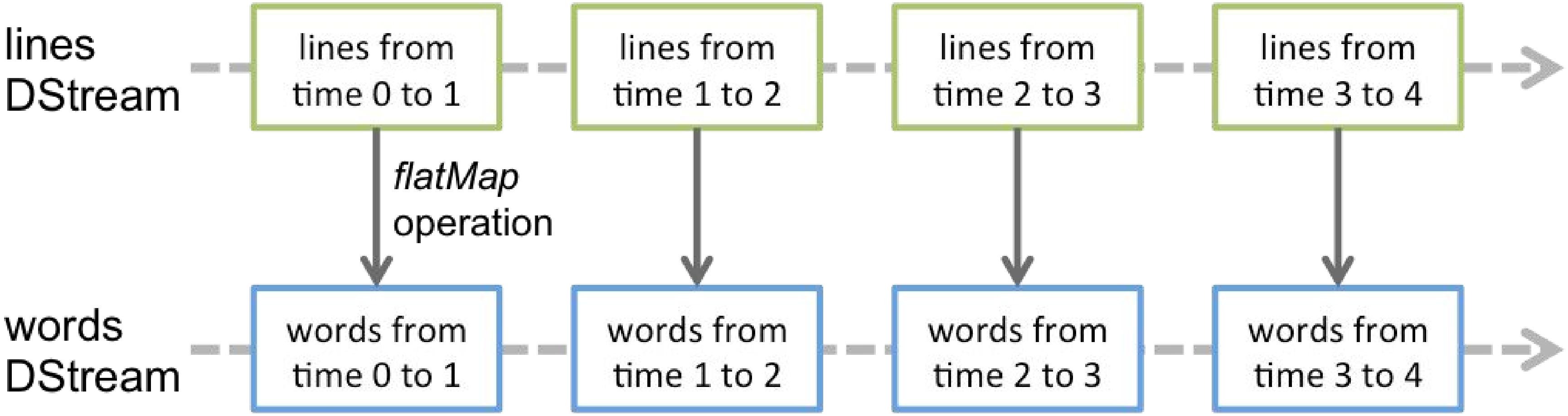
lines from time 0 to 1

lines from time 1 to 2

lines from time 2 to 3

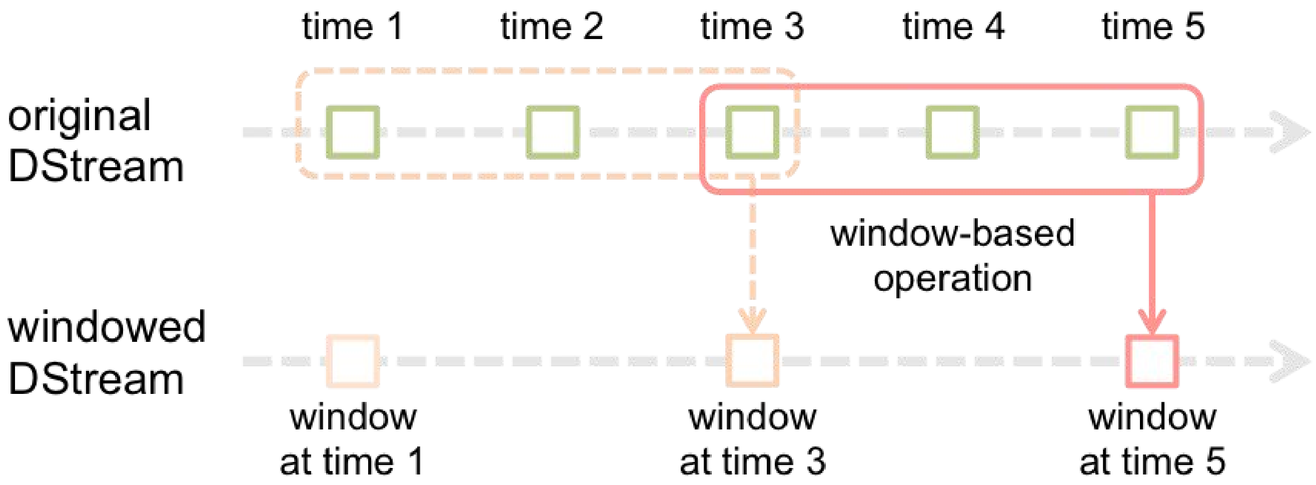
lines from time 3 to 4

DStreams


lines DStream
lines from time 0 to 1
lines from time 1 to 2
lines from time 2 to 3
lines from time 3 to 4
flatMap operation
words DStream
words from time 0 to 1
words from time 1 to 2
words from time 2 to 3
words from time 3 to 4

DStrems - windowed computations

- You provide windows length, and sliding interval
- Combined RDD per window, operations on that RDD



Summary

- Spark Streaming can use pre-defined or custom data receiver
- Streams are converted to batches of requested time interval
    - Which might further divide as microbatches
- Each batch is a DStream
- Each DStream has corresponding set of RDDs
- Each DStream operation causes a set of RDD transformations
- Utilize windowed computation given window length and sliding interval

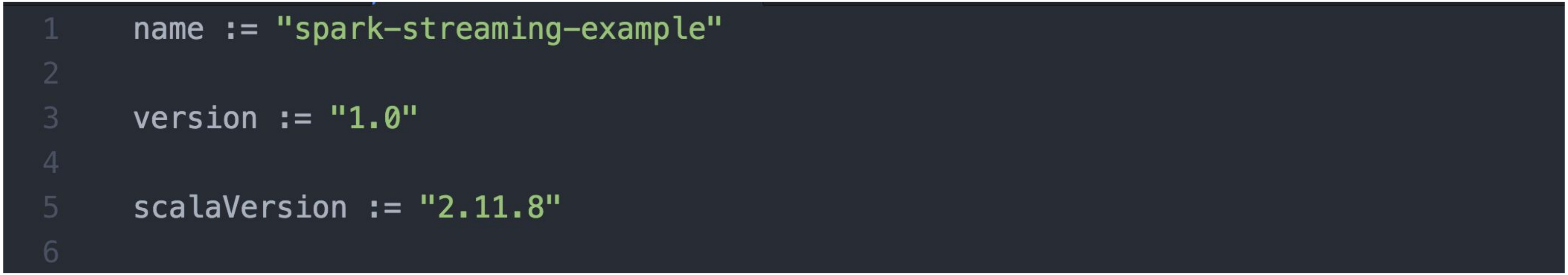
Let’s write some code!

Supergloo / Todd McGrath’s Slack streaming example with minor modifications

Project setup

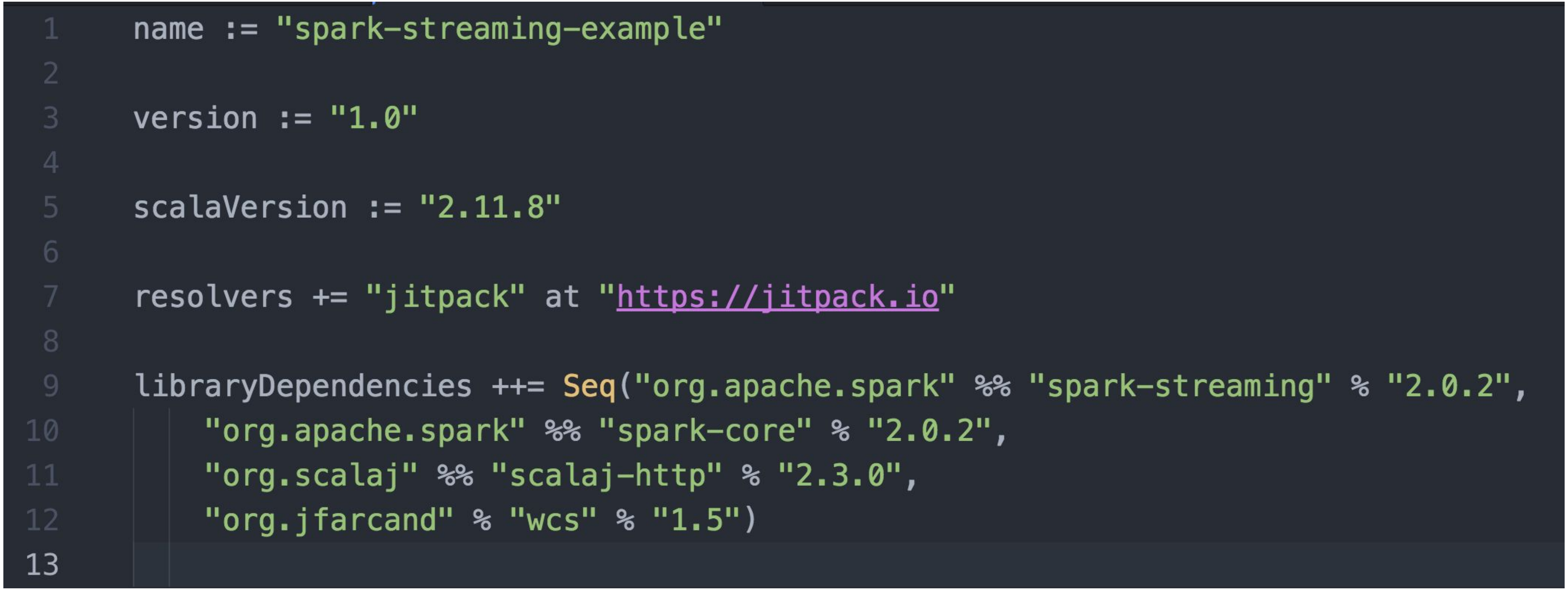
```
mkdir -p spark-streaming-example/src/main/scala/com/slacky

cd spark-streaming-example
```

build.sbt



```
name := "spark-streaming-example"

version := "1.0"

scalaVersion := "2.11.8"

```

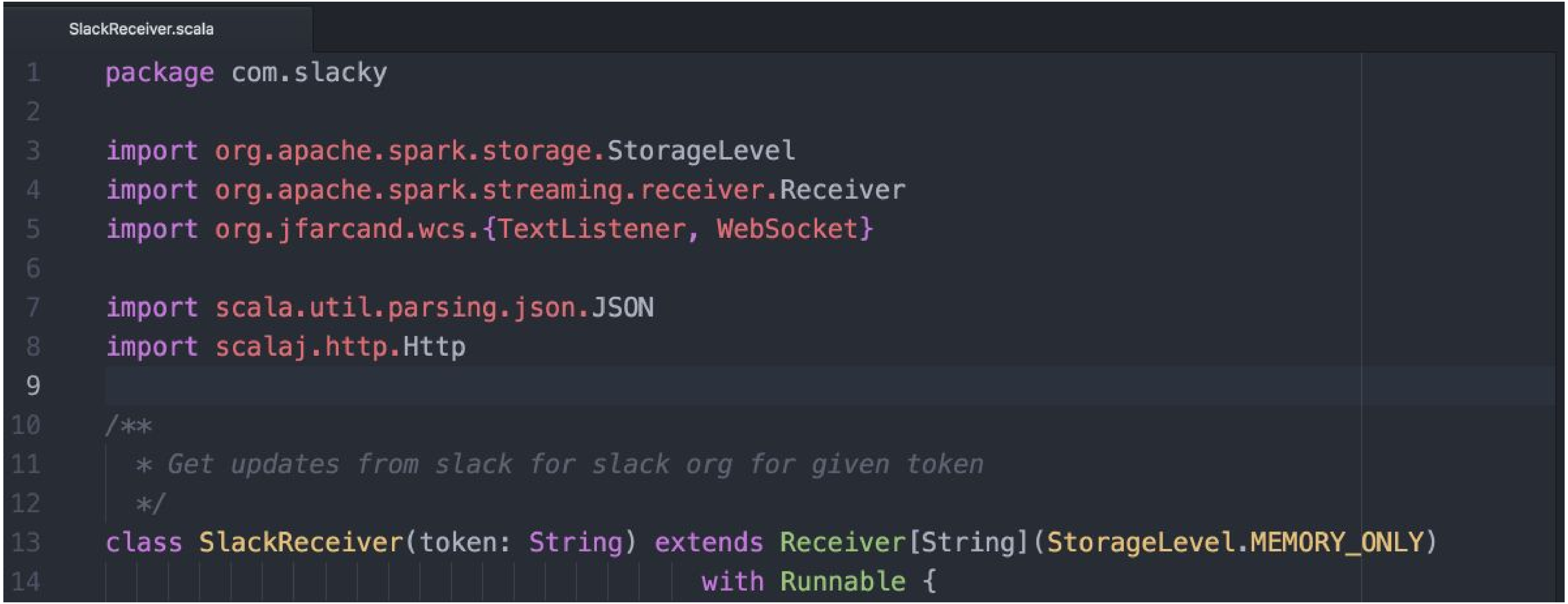
build.sbt



```
name := "spark-streaming-example"

version := "1.0"

scalaVersion := "2.11.8"

resolvers += "jitpack" at "https://jitpack.io"

libraryDependencies ++= Seq("org.apache.spark" %% "spark-streaming" % "2.0.2",
    "org.apache.spark" %% "spark-core" % "2.0.2",
    "org.scalaj" %% "scalaj-http" % "2.3.0",
    "org.jfarcand" % "wcs" % "1.5")

```

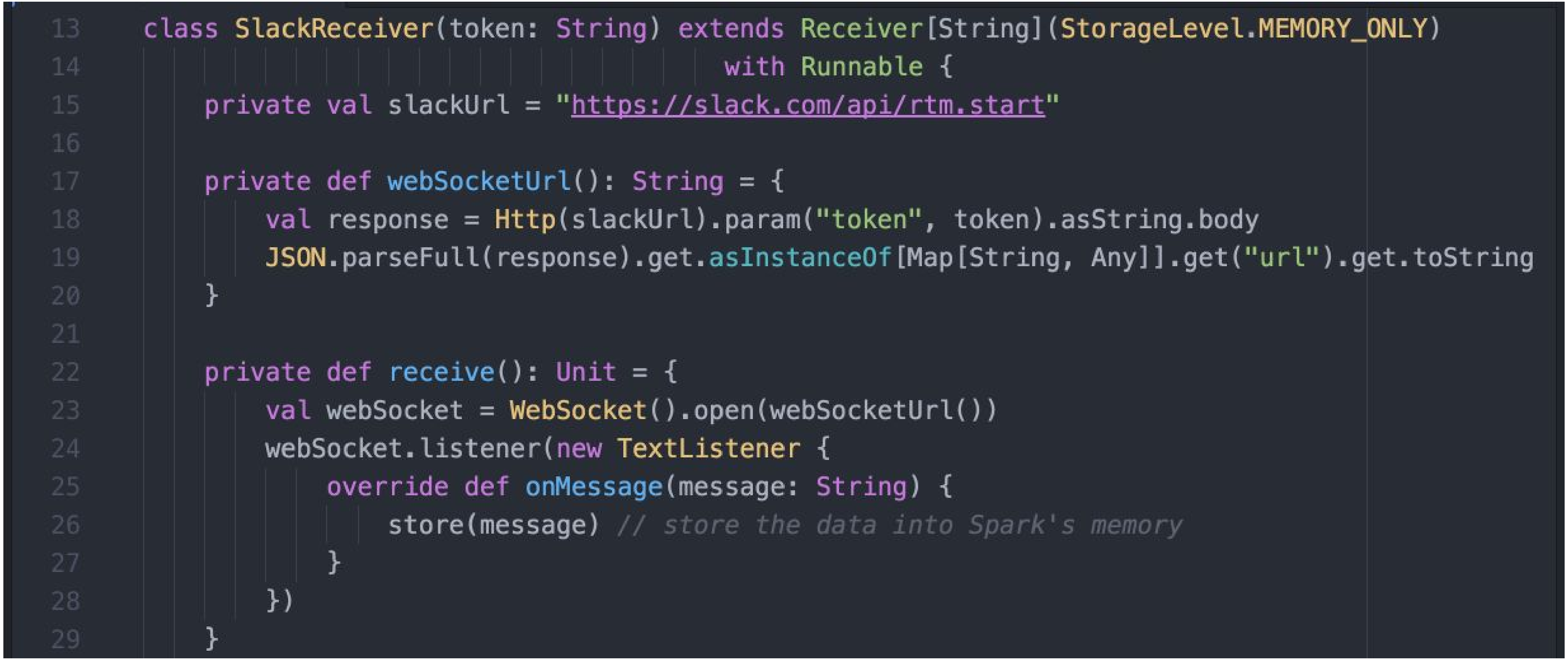
```
src/main/scala/com/slacky/SlackReceiver.scala
```

src/main/scala/com/slacky/SlackReceiver.scala



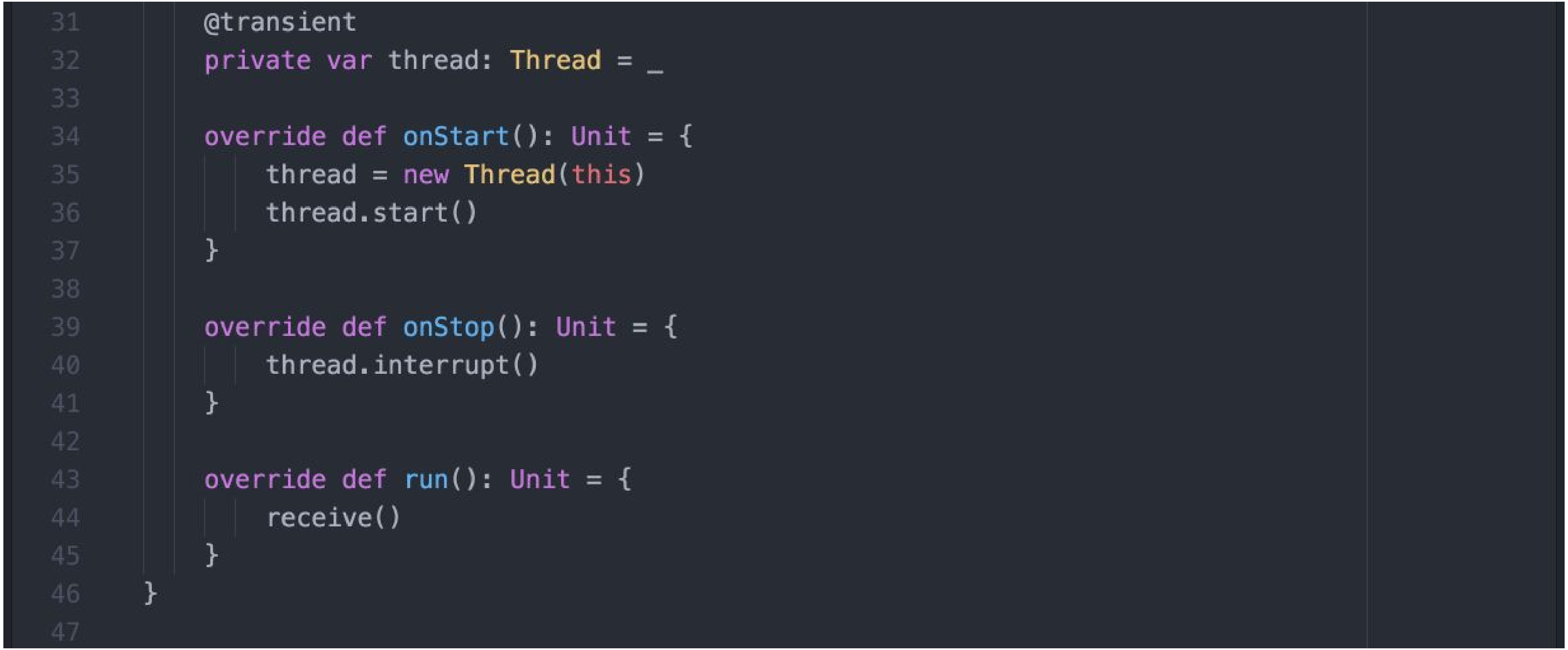
```scala
package com.slacky

import org.apache.spark.storage.StorageLevel
import org.apache.spark.streaming.receiver.Receiver
import org.jfarcand.wcs.{TextListener, WebSocket}

import scala.util.parsing.json.JSON
import scalaj.http.Http

/**
  * Get updates from slack for slack org for given token
  */
class SlackReceiver(token: String) extends Receiver[String](StorageLevel.MEMORY_ONLY)
                                      with Runnable {
```

src/main/scala/com/slacky/SlackReceiver.scala



```scala
class SlackReceiver(token: String) extends Receiver[String](StorageLevel.MEMORY_ONLY)
                   with Runnable {
  private val slackUrl = "https://slack.com/api/rtm.start"

  private def webSocketUrl(): String = {
    val response = Http(slackUrl).param("token", token).asString.body
    JSON.parseFull(response).get.asInstanceOf[Map[String, Any]].get("url").get.toString
  }

  private def receive(): Unit = {
    val webSocket = WebSocket().open(webSocketUrl())
    webSocket.listener(new TextListener {
      override def onMessage(message: String) {
        store(message) // store the data into Spark's memory
      }
    })
  }
```

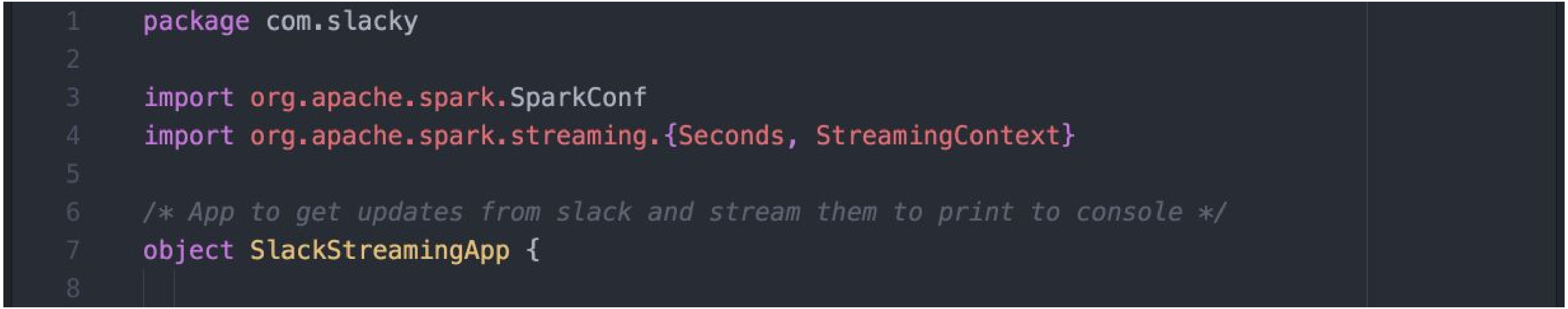
src/main/scala/com/slacky/SlackReceiver.scala



```scala
    @transient
    private var thread: Thread = _

    override def onStart(): Unit = {
        thread = new Thread(this)
        thread.start()
    }

    override def onStop(): Unit = {
        thread.interrupt()
    }

    override def run(): Unit = {
        receive()
    }
}

```

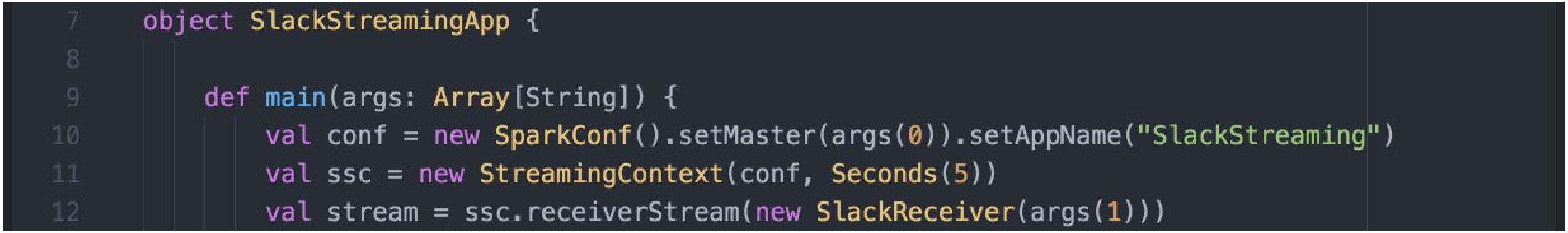
src/main/scala/com/slacky/SlackStreamingApp.scala

src/main/scala/com/slacky/SlackStreamingApp.scala



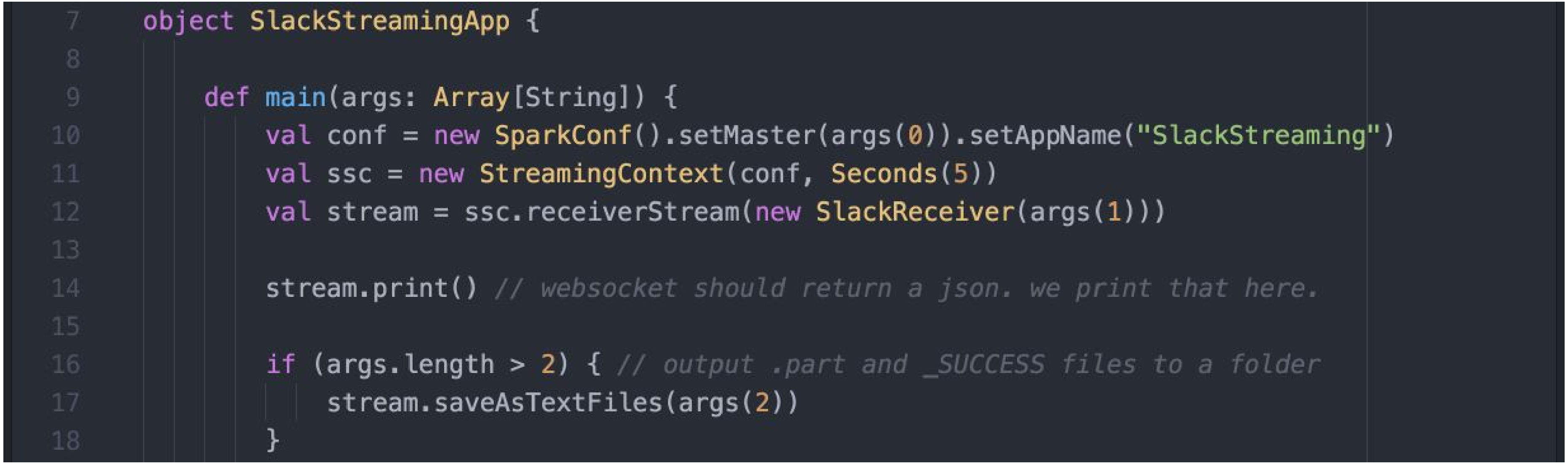
```scala
package com.slacky

import org.apache.spark.SparkConf
import org.apache.spark.streaming.{Seconds, StreamingContext}

/* App to get updates from slack and stream them to print to console */
object SlackStreamingApp {

```

src/main/scala/com/slacky/SlackStreamingApp.scala



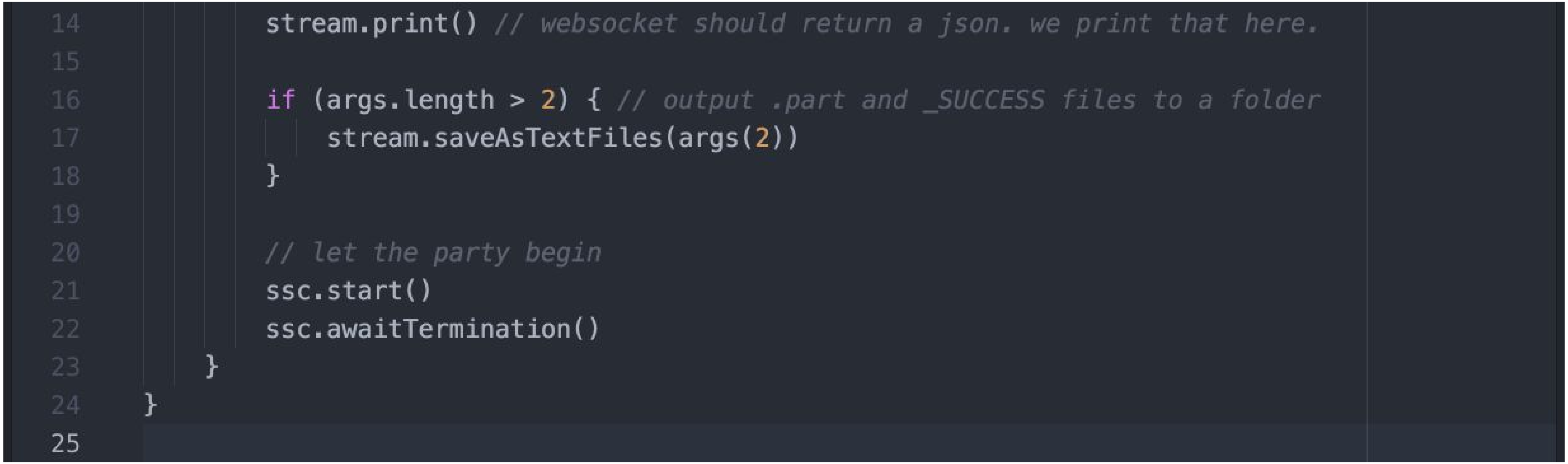
```scala
object SlackStreamingApp {

    def main(args: Array[String]) {
        val conf = new SparkConf().setMaster(args(0)).setAppName("SlackStreaming")
        val ssc = new StreamingContext(conf, Seconds(5))
        val stream = ssc.receiverStream(new SlackReceiver(args(1)))
```

src/main/scala/com/slacky/SlackStreamingApp.scala



```scala
object SlackStreamingApp {

    def main(args: Array[String]) {
        val conf = new SparkConf().setMaster(args(0)).setAppName("SlackStreaming")
        val ssc = new StreamingContext(conf, Seconds(5))
        val stream = ssc.receiverStream(new SlackReceiver(args(1)))

        stream.print() // websocket should return a json. we print that here.

        if (args.length > 2) { // output .part and _SUCCESS files to a folder
            stream.saveAsTextFiles(args(2))
        }
```

src/main/scala/com/slacky/SlackStreamingApp.scala



```scala
        stream.print() // websocket should return a json. we print that here.

        if (args.length > 2) { // output .part and _SUCCESS files to a folder
            stream.saveAsTextFiles(args(2))
        }

        // let the party begin
        ssc.start()
        ssc.awaitTermination()
    }
}

```

Prep local spark

```
# go to your spark install directory

$   sbin/start-master.sh

# confirm master is on

$   sbin/start-slave.sh
spark://dmistry-ltm.internal.salesforce.com:7077

# confirm slave is on
```

Stream all of the things!

```
# go to your root directory of example project

$ sbt

sbt> run local[5] <slack_token> output

# and now we wait :)
```

Homework

Instead of printing messages to console,

(1) Grab lines from Titanic dataset, and stream them to spark. 1 rec per 10sec.
    (a) Directly as csv, or through slack, or kafka if you're feeling adventurous
(2) Report running count and/or mean of surviving vs dead people, and their ages

More homework, if you can't contain your excitement!

- Remove survivorship info from input data.
- Utilize your MLlib model from earlier exercise to predict the survivorship as the records arrive.
- Report the output to a csv file.